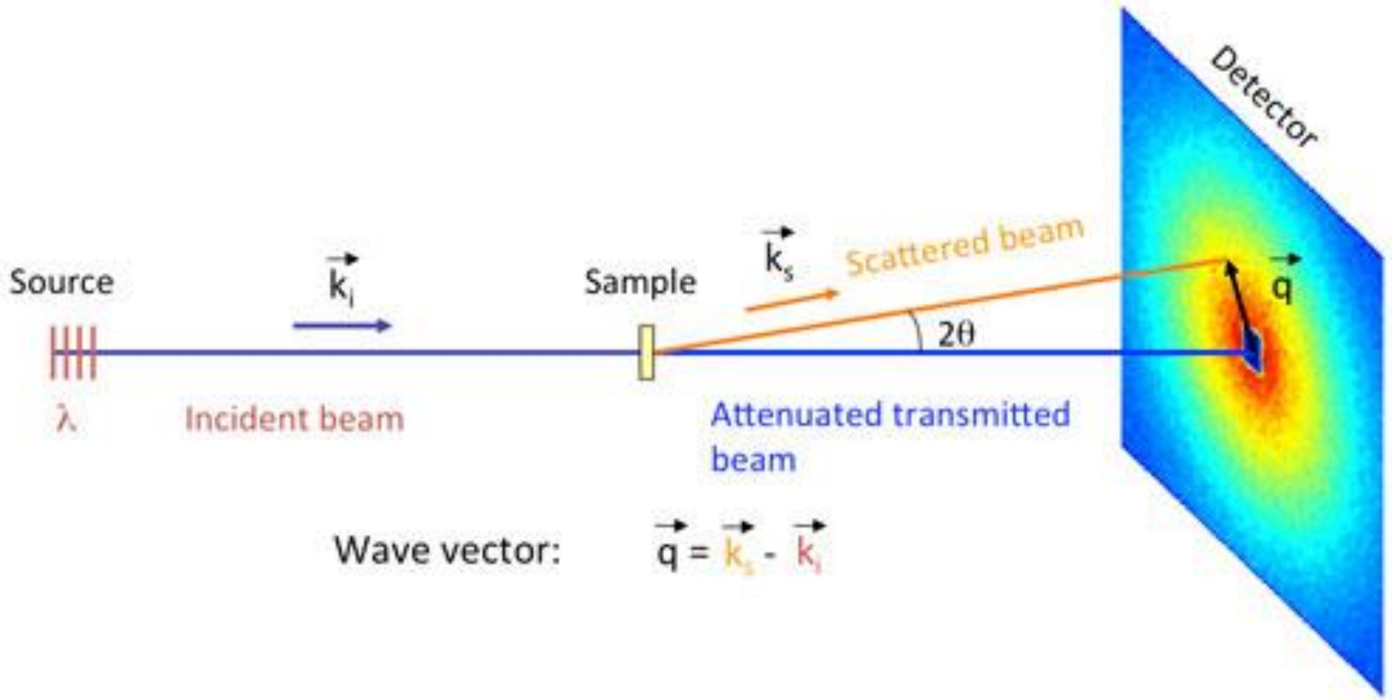
GPU PROGRAMMING: 2D FITTING FOR SMALL-ANGLE NEUTRON SCATTERING DATA

Helen Park

Paul Kienzle

Matt Wasbrough

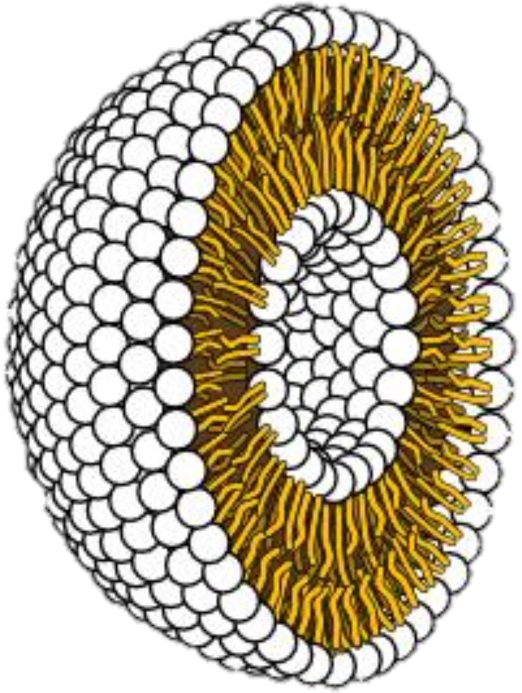
DATA USED: COLLECTED BY NEUTRON SCATTERING



- Neutrons scatter off large scale structures in the material
- Reveal structure of material
- Colors refer to intensity/ density of neutrons scattered

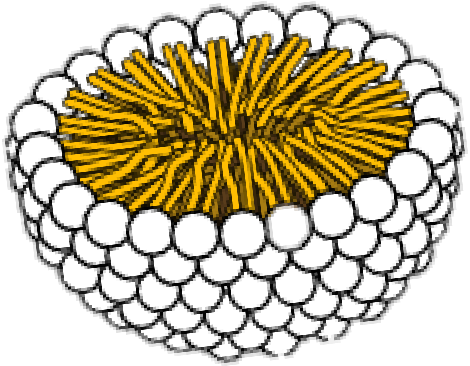
SURFACTANT DATA

- Has a hydrophobic tail, hydrophilic head
- Form variety of shapes
- Used in soap, drug delivery, even mayonnaise!

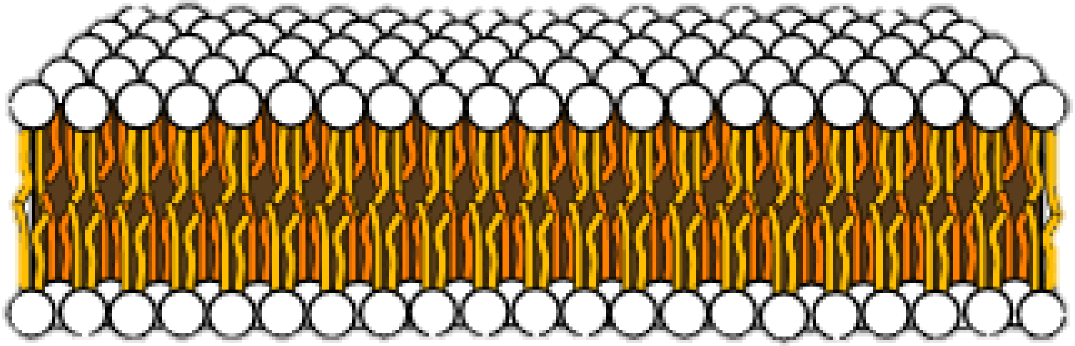


Liposome

Micelle

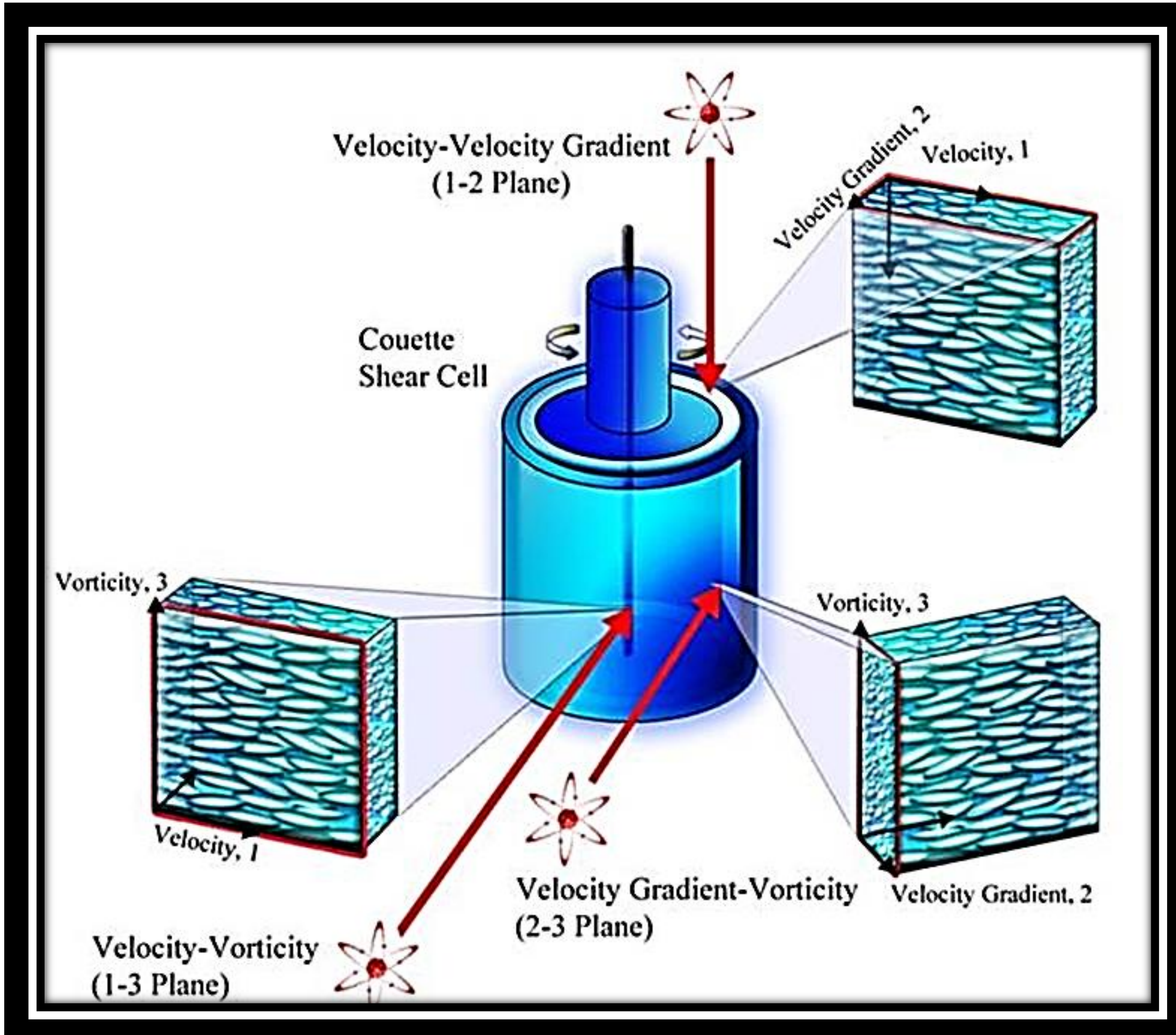


Bilayer Sheet/Lamellar



UNDER SHEAR STRESS

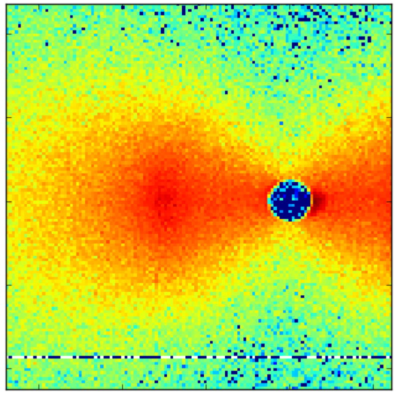
- Device spins the surfactants at different speeds
- Surfactants change shape
- How does the geometry of the material change under different frequencies?



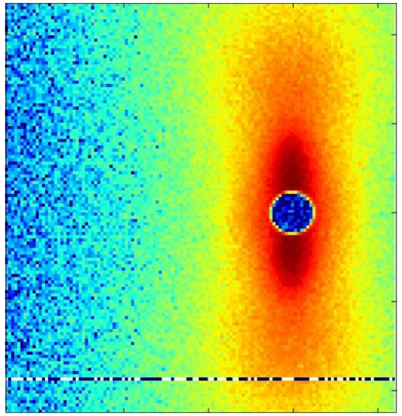
2D FITTING

- Fitting allows a better understanding of the data
- Chose a model (such as Core-Shell-Cylinder, Lamellar, Triaxial-Ellipse) to best describe the phase of the surfactant micelles
- Manipulate variables to fit the shape and orientation of the data

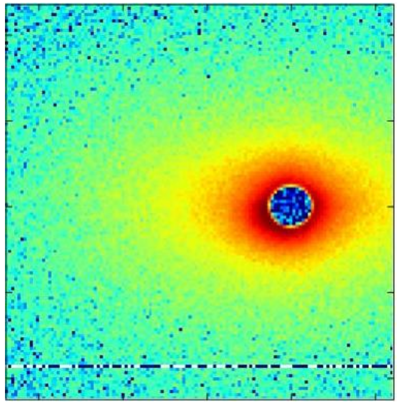
➢ These are examples of data collected under various amounts of shear stress



0 Hz

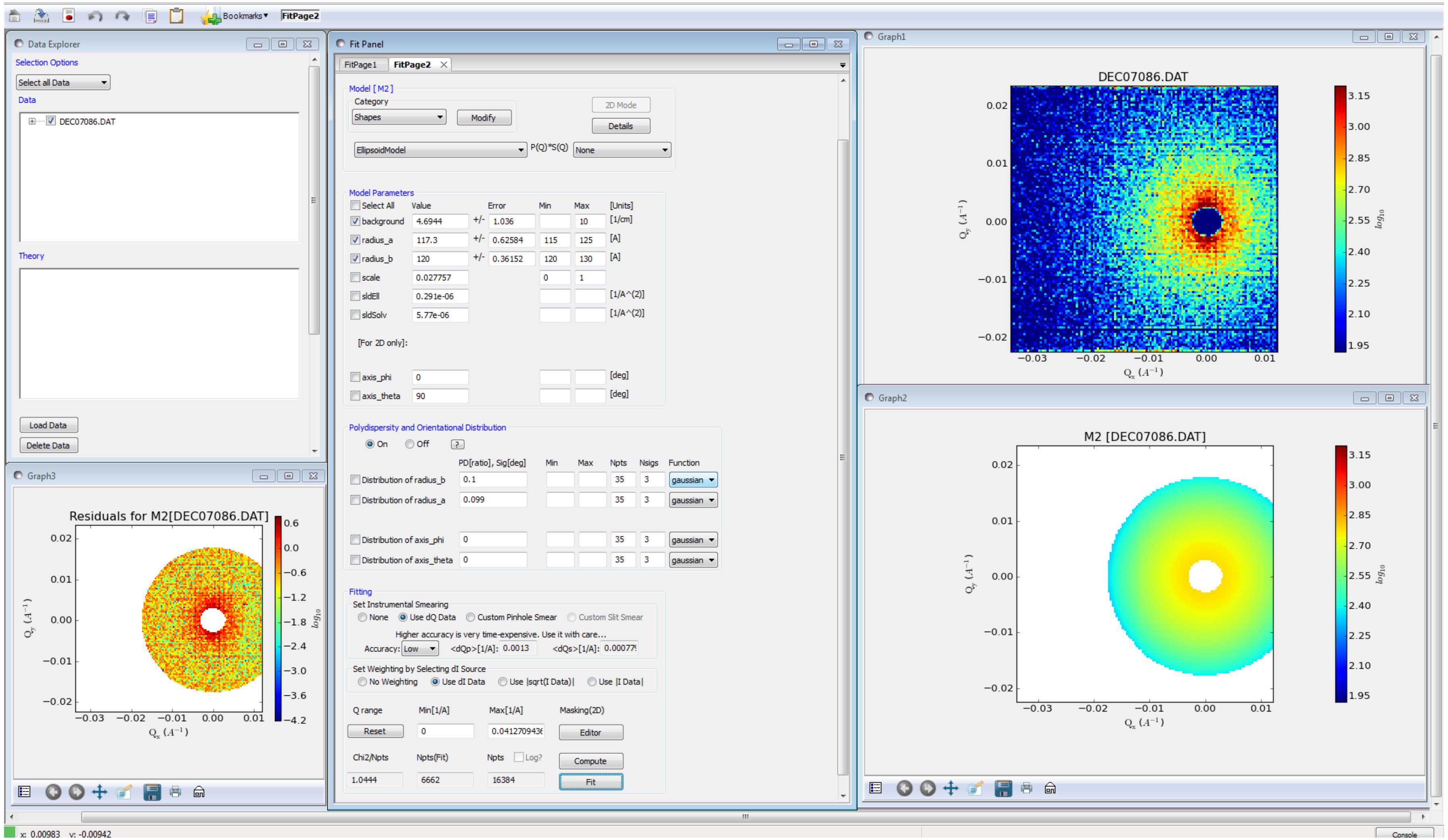


1230 Hz



7000 Hz

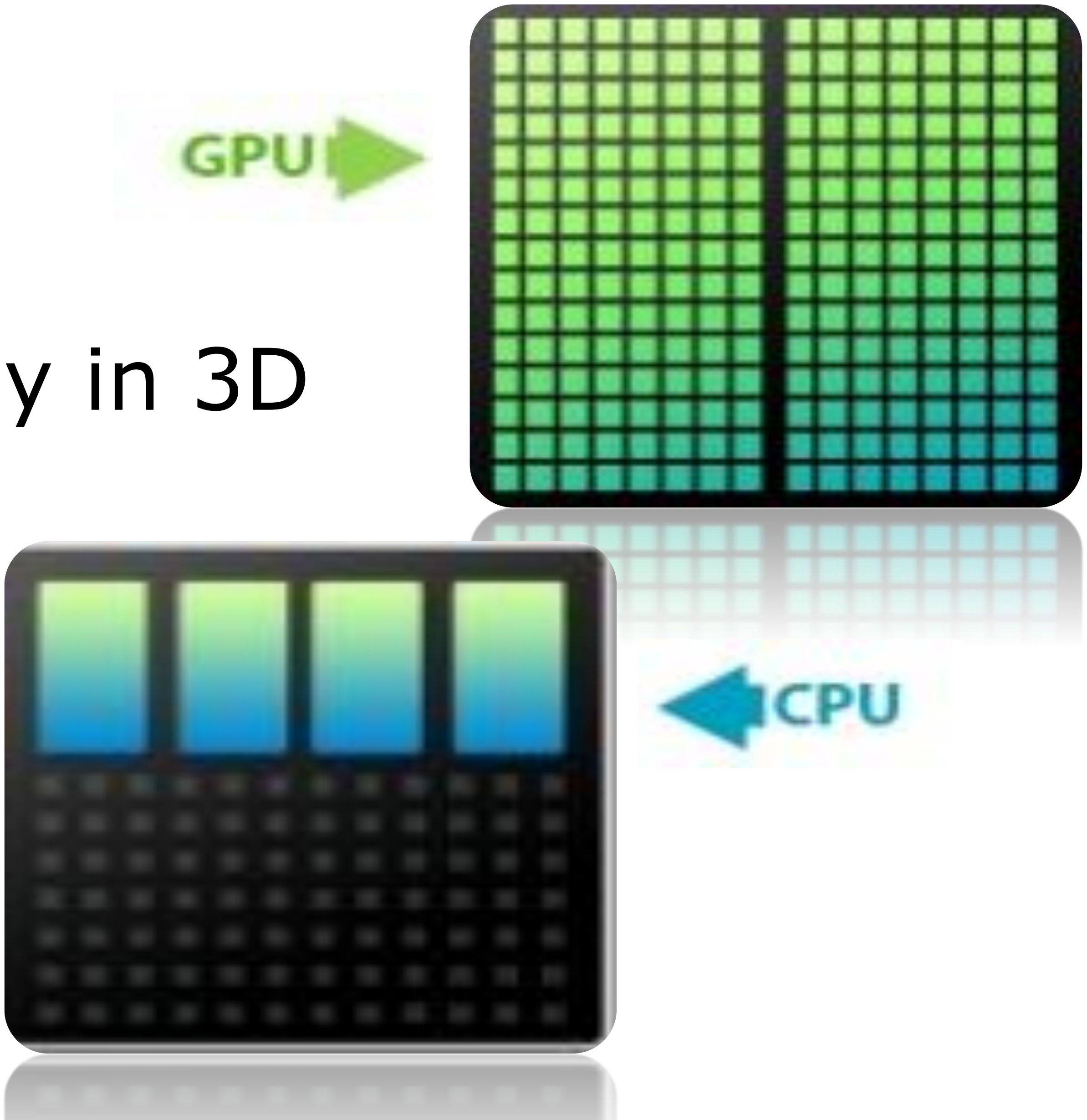
SASVIEW: FITS THE DATA

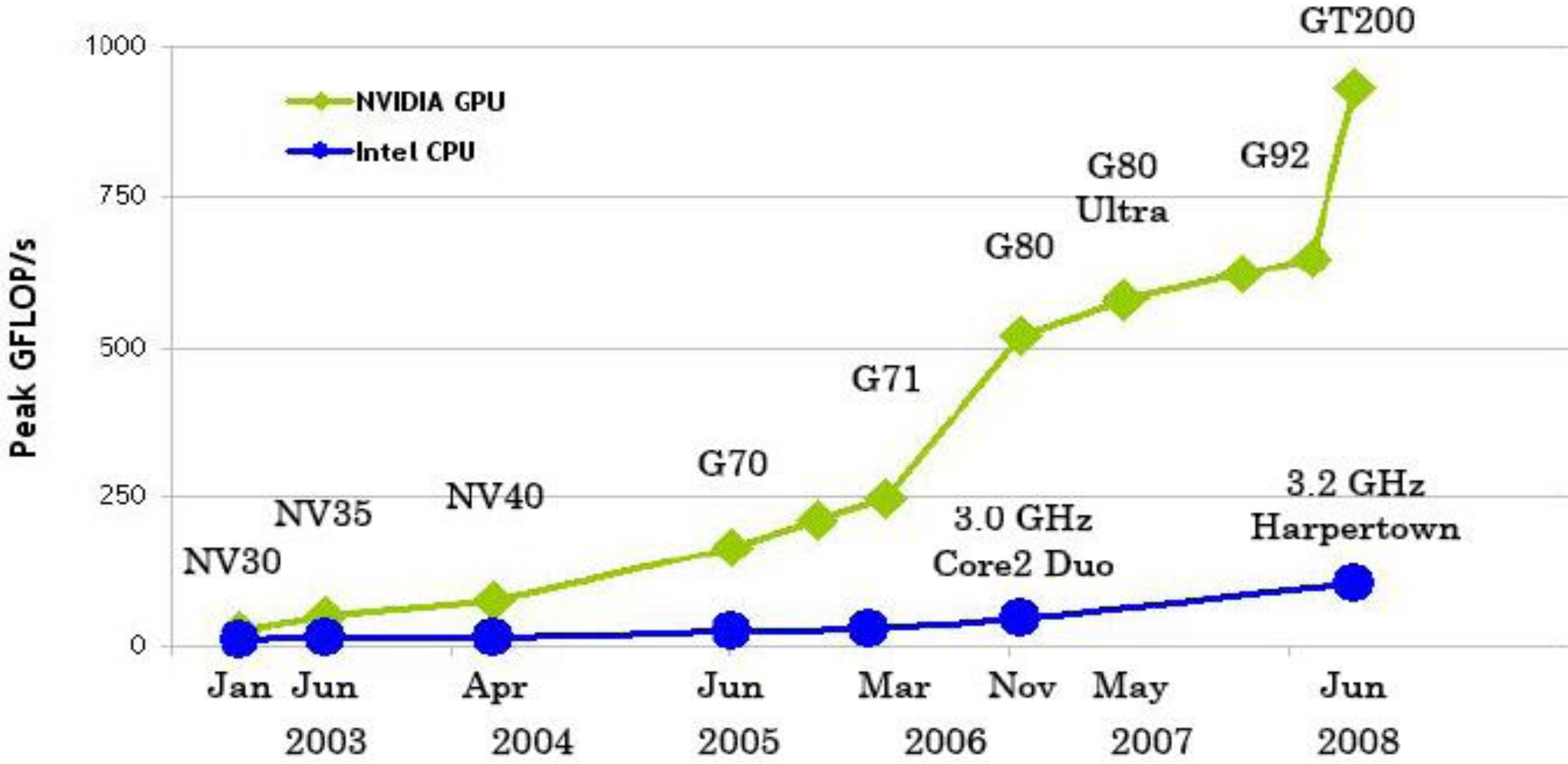


- Choose model that best describes data
- Issues: Slow, crashes often

SOLUTION!

- GPU programming: faster
- CPU—found in most PCs
- GPU—previously used solely in 3D gaming
- GPU allows parallel processing, 1000s of threads, 100s more cores than CPU





GT200
1000
750
500
250
0
NVIDIA GPU
Intel CPU
Peak GFLOP/s
G92
G80
Ultra
G80
G71
G70
NV40
NV35
NV30
3.2 GHz
Harpertown
3.0 GHz
Core2 Duo
Jan
Jun
Apr
Jun
Mar
Nov
May
Jun
2003
2004
2005
2006
2007
2008

- GPUs are cheap, fast, and energy efficient

	Processor	US$	Cores	GHz	GFlops	GFlops/W
CPU	Intel i7 x6	1000	6	3.3	100	<1
	AMD Phenom II x6	300	6	4.0	100	<1
	NVidia GeForce 480	500	480	0.7	1400	5
GPU	ATI Radeon 5970	700	3200	0.7	4600	15

Speed-ups of different projects

MA Hospital	300X
U Rochester	160X
U Amsterdam	150X
Harvard	130X
U Pennsylvania	130X
Nanyang Tech	130X
U Illinois	125X
Cambridge U	100X
Boise State	100X
Florida U	100X

- For matrix multiplication: **GPU = 150*CPU MFLOPS** (Mega floating point operations per second)

- **But**, need to tune algorithm and memory transfer for every sized GPU, and for each kernel and program

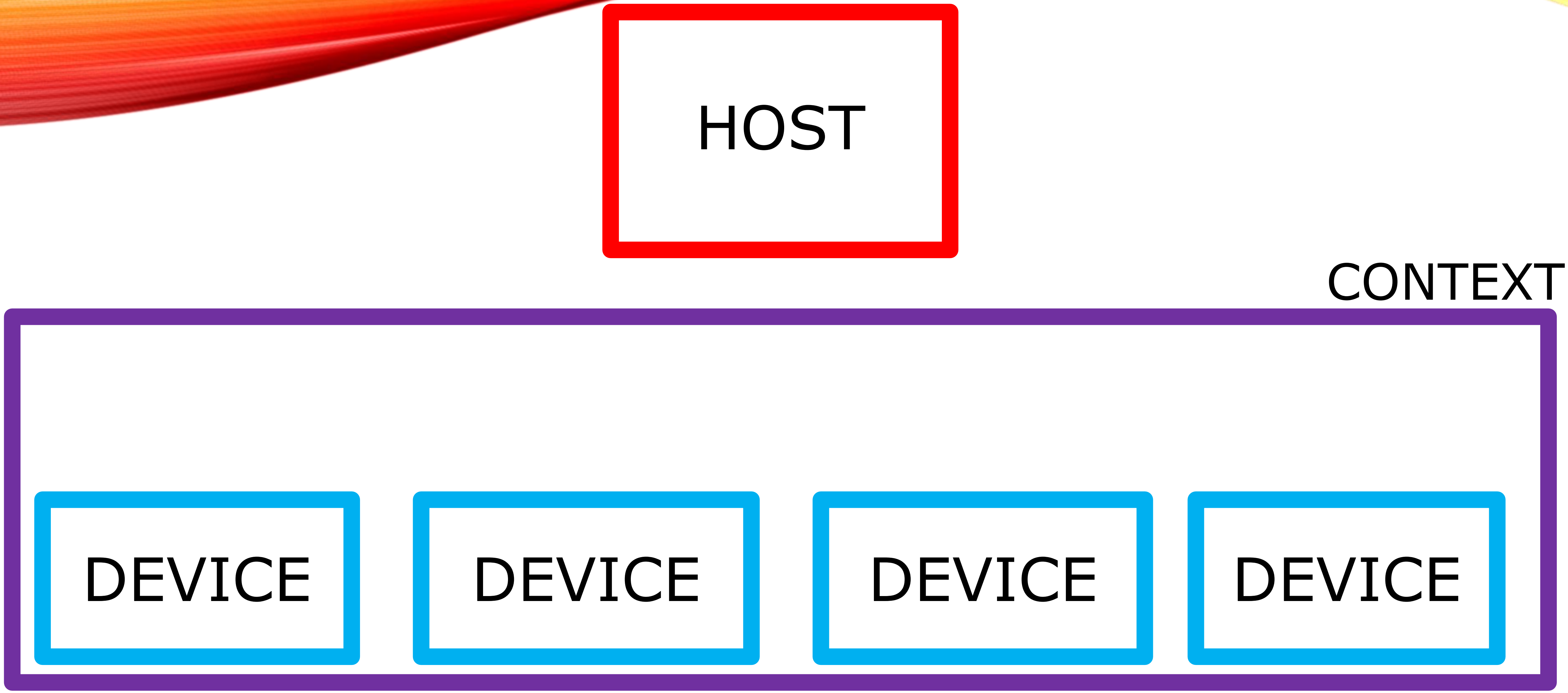
HOST

The Host (CPU) passes memory buffers, kernels, and queue commands to the device, and receives the result, also in a buffer

HOST

CONTEXT

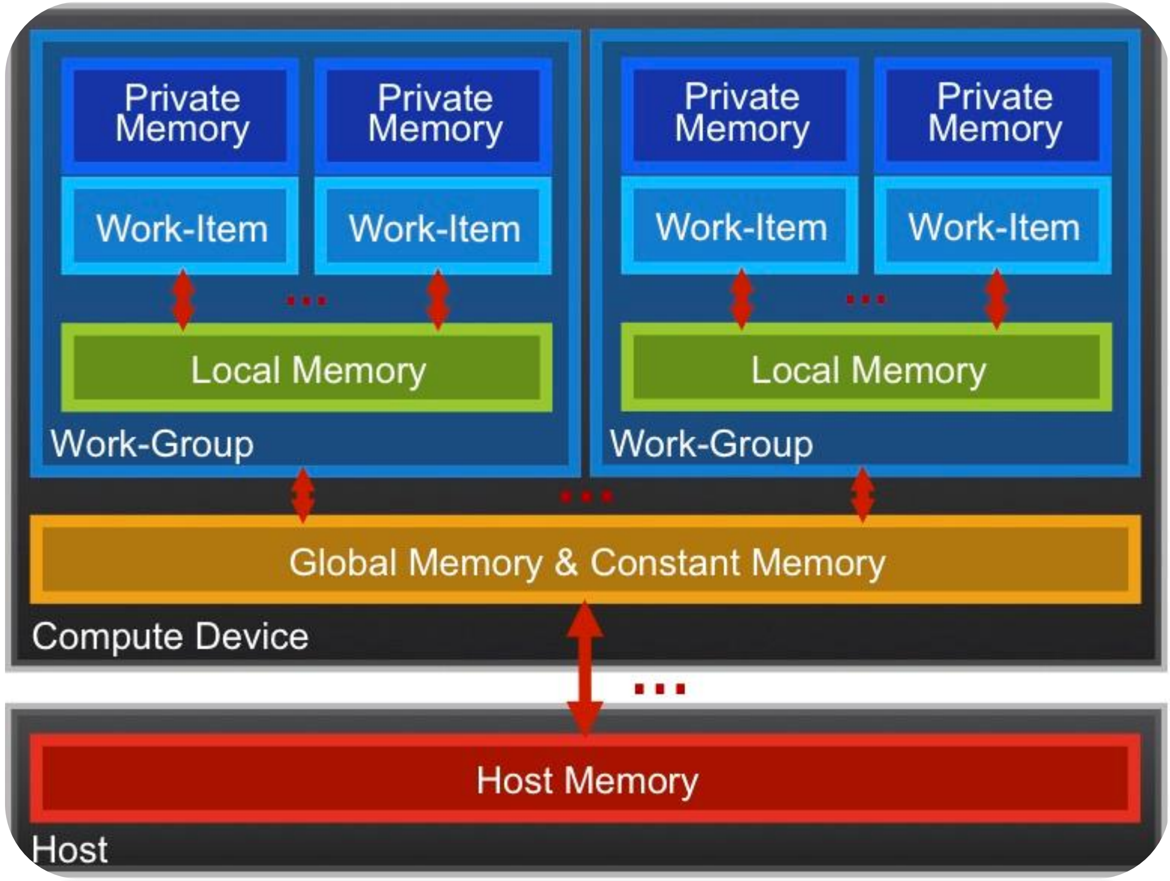
The Context holds the GPU(s)—varying shaped & sizes



A Device (GPU) handles the computations.

Handling of memory transfer global → local → private effect speeds

A Device has global, local, and private memory, and many work groups that perform calculations in parallel

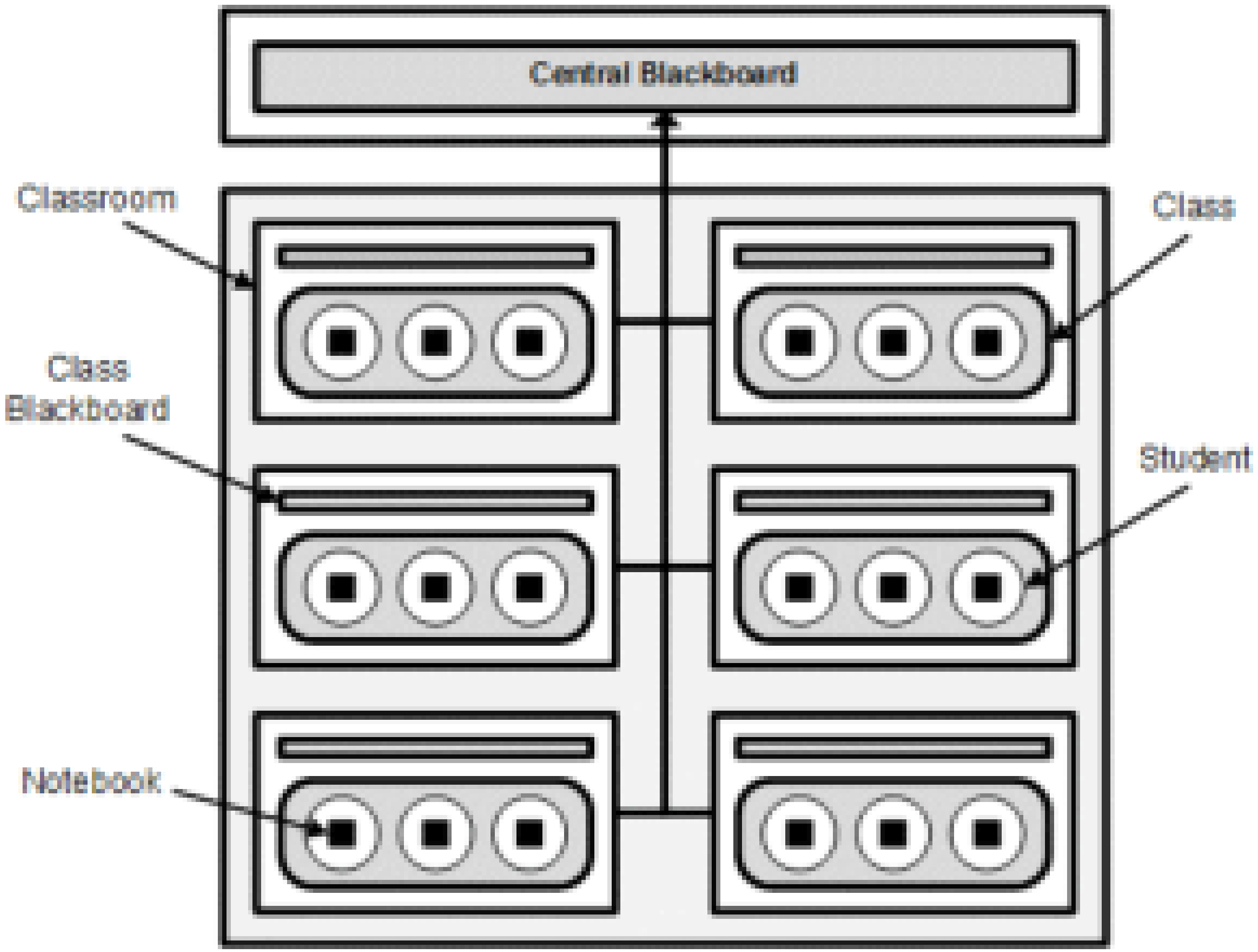


OPENCL DESIGN

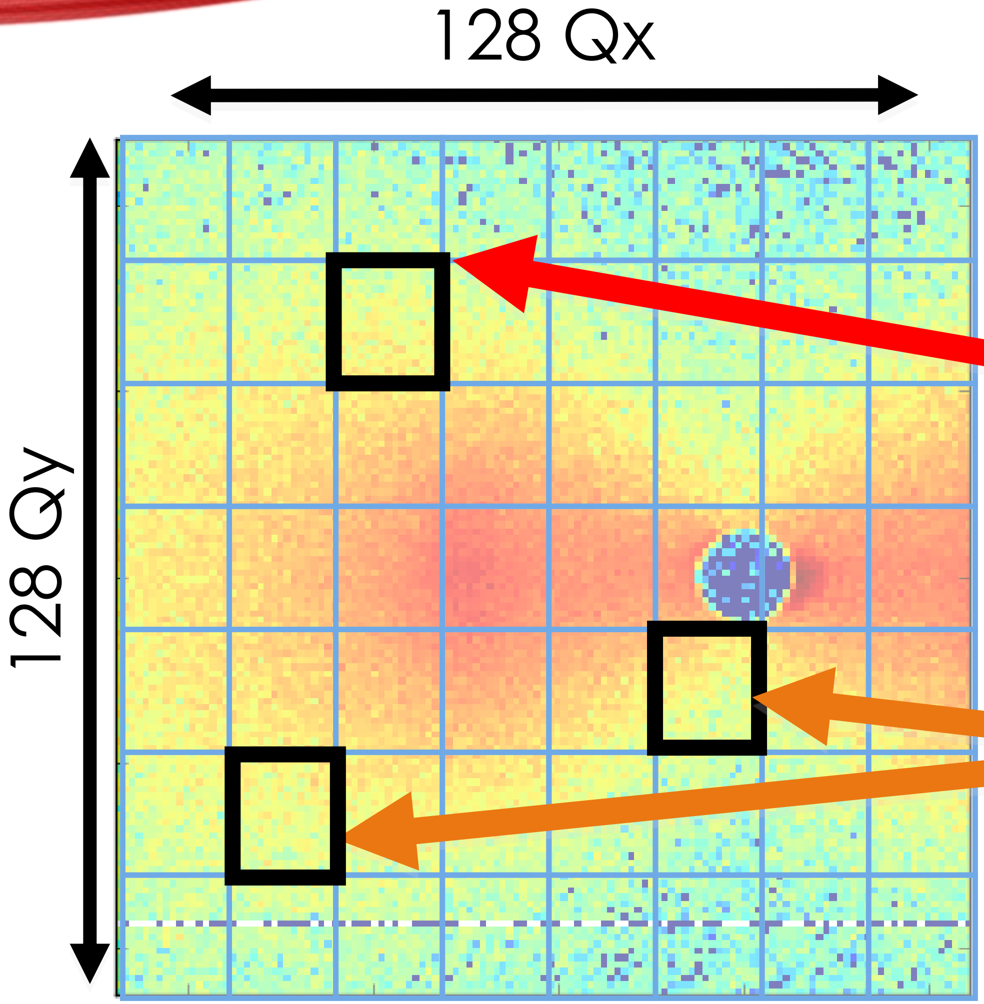
Central Blackboard: Global Memory

Class Blackboard: Local Memory

Notebook: Private Memory

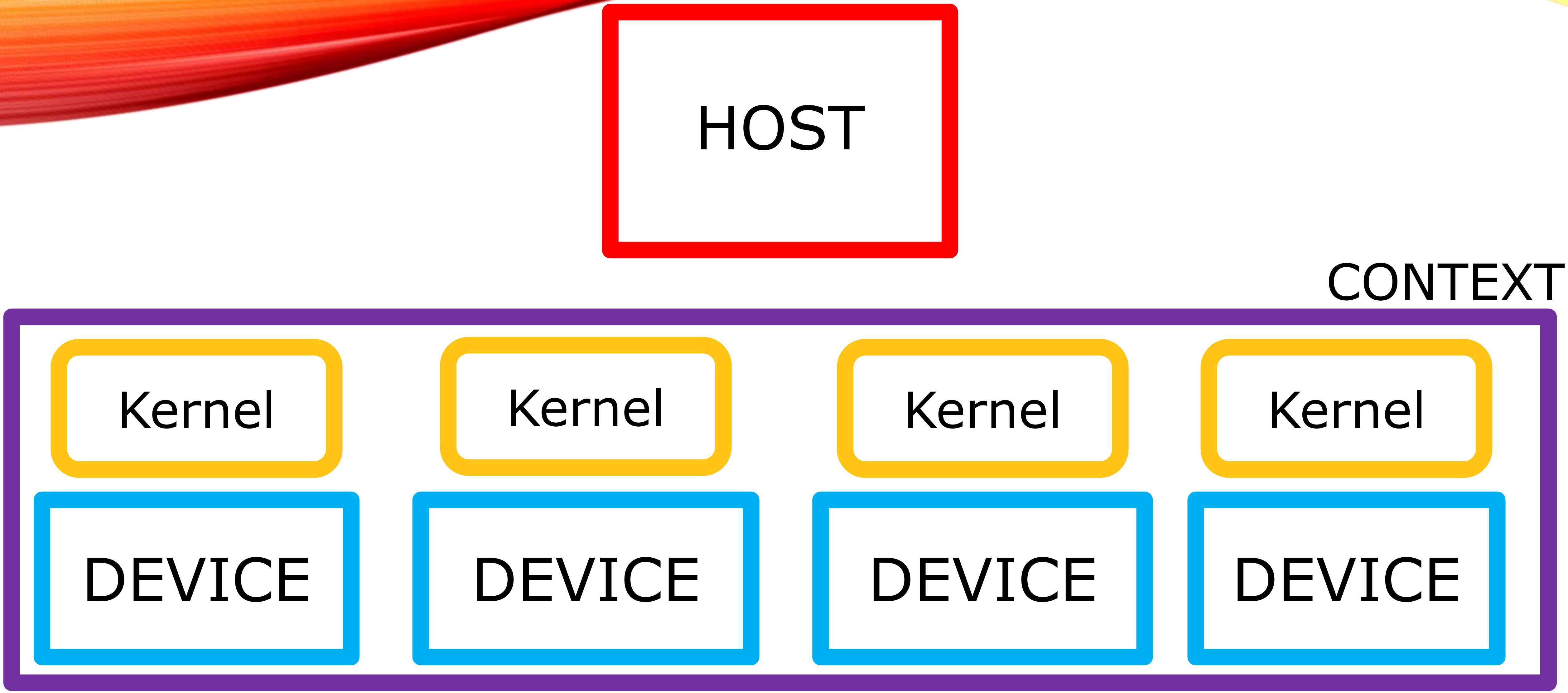


DEVICE ARCHITECTURE

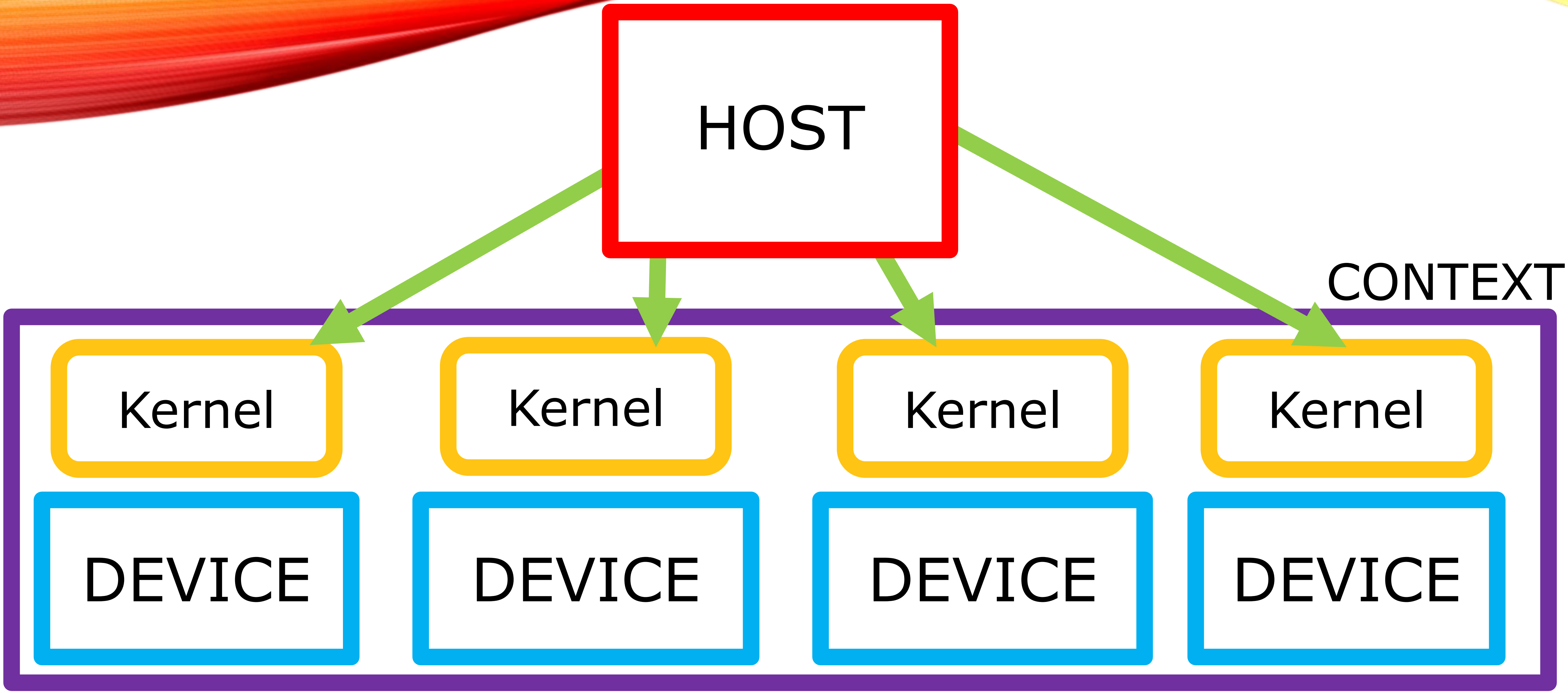


A work group, 128x128 performs computations

Work-groups run in parallel, do not interact



The Kernel is the physical code, or program, that are computed by the device



The Queue relays and manages how kernels and buffers are translated and organized

Our Kernel: calculates scattering intensity, stored in a 2D array

- In our algorithm, work-group takes random Qx &Qy and calculates the scattering density at that point
- Performs this until every Qx & Qy complete
- Adds results and returns to CPU

Cylinder Model equations:

$$P(q,\alpha) = \frac{scale}{V_s} f^2(q) + bkg$$

$$f(q) = 2(\rho_c - \rho_s)V_c \sin[qL\cos\alpha/2]\,/[qL\cos\alpha/2]\frac{J_1[qr\sin\alpha]}{[qr\sin\alpha]}$$

$$+2(\rho_s - \rho_{solv})V_s \sin[q(L+2t)\cos\alpha/2]/\,[q(L+2t)\cos\alpha/2]\frac{J_1[q(r+t)\sin\alpha]}{[q(r+t)\sin\alpha]}$$

$$V_s = \pi(R+t)^2 \cdot (L+2t)$$

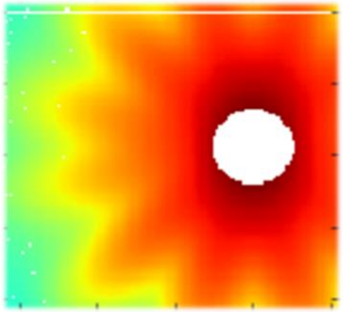
Cylinder Model kernel:

```
__kernel void CylinderKernel(__global const real *qx, global const real *qy, __global real *_ptvalue, cons
const real rr, const real h, const real scale, const real radius_weight, const real length_weight,
const real theta_weight, const real phi_weight, const real cyl_theta,
const real cyl_phi, const int count, const int size)
{
    // qq is the q-value for the calculation (1/A)
    // rr is the radius of the cylinder (A)
    // h is the -LENGTH of the cylinder = L (A)
    int i = get_global_id(0);

    if(i < count)
    {
        real be=0.0; real si=0.0;
        real qq = sqrt(qx[i]*qx[i]+qy[i]*qy[i]);

        real pi = 4.0*atan(1.0);
        real theta = cyl_theta*pi/180.0;
        real cos_val = cos(theta)*cos(cyl_phi*pi/180.0)*(qx[i]/qq) + sin(theta)*(qy[i]/qq);

        real alpha = acos(cos_val);
        if(alpha == 0.0){
            alpha = 1.0e-26;
        }
        real besarg = qq*rr*sin(alpha);
        real siarg = qq*h/2*cos(alpha);

        real bj = NR_BessJ1(besarg);
        real d1 = qq*rr*sin(alpha);

        if (besarg == 0.0){
            be = sin(alpha);
        }
        else{
            be = bj*bj*4.0*sin(alpha)/(d1*d1);
        }
        if(siarg == 0.0){
            si = 1.0;
        }else{
            si = sin(siarg)*sin(siarg)/(siarg*siarg);
        }
        real form = be*si/sin(alpha);
        real answer = sub*sub*form*acos(-1.0)*rr*rr*h*1.0e8*scale;

        _ptvalue[i] = radius_weight*length_weight*theta_weight*phi_weight*answer*pow(rr,2)*h;
       // if (size>1) {
```

POLYDISPERSITY

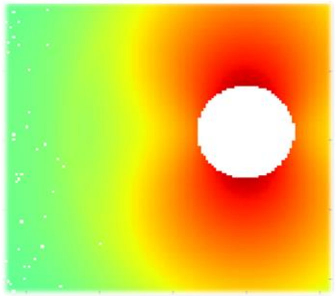
Loop for polydispersity in CPU

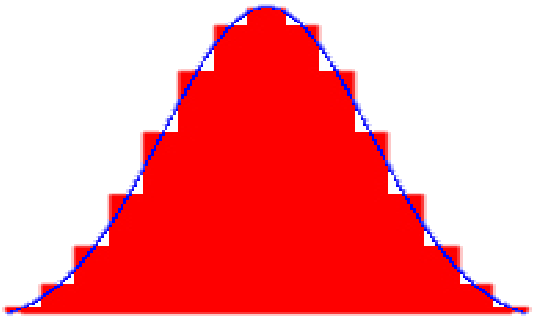
- Size of polydispersity corresponds to width of bell-curve
- Allow a variety of values for a variable (like length)
- For example, high polydispersity in theta gives a larger range of angles
- Also, the more bins, the more accurate the fit



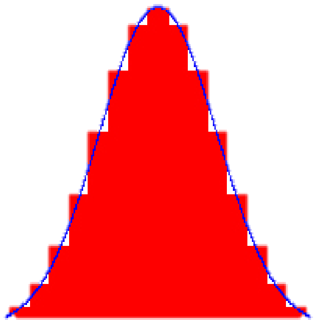
← High polydispersity, but low number of bins (5)

→ Lower polydispersity, but many bins (40)





Cost: higher bins means much slower fit, so need to balance



FITTING PROGRAM

- In CPU, program the context, device(s), the queue to relay information, and write the buffers for variables, and return values
- Using bumps, loop (again!) over the entire program to fit different variables

```
model = SasModel(data, GpuCylinder,
scale=0.0104,
radius=92.5,
length=798.3,
sldCyl=.29e-6,
sldSolv=7.105e-6,
background=5,
cyl_theta=0,
cyl_phi=0,
cyl_theta_pd=22.11,
cyl_theta_pd_n=20,
cyl_theta_pd_nsigma=3,
radius_pd=.0084,
radius_pd_n=10,
radius_pd_nsigma=3,
length_pd=0.493,
length_pd_n=10,
length_pd_nsigma=3,
cyl_phi_pd=0,
cyl_phi_pd_n=1,
cyl_phi_pd_nsigma=3,
dtype='f
```

```
# SET THE FITTING PARAMETERS
model.radius.range(1, 500)
model.length.range(1, 4000)
model.cyl_theta.range(-90,100)
model.cyl_theta_pd.range(0, 90)
model.cyl_theta_pd_n = model.cyl_theta_pd + 5
model.radius_pd.range(0, 90)
model.length_pd.range(0, 90)
model.scale.range(0, 1)
model.background.range(0, 100)
model.sldCyl.range(0, 1)
```

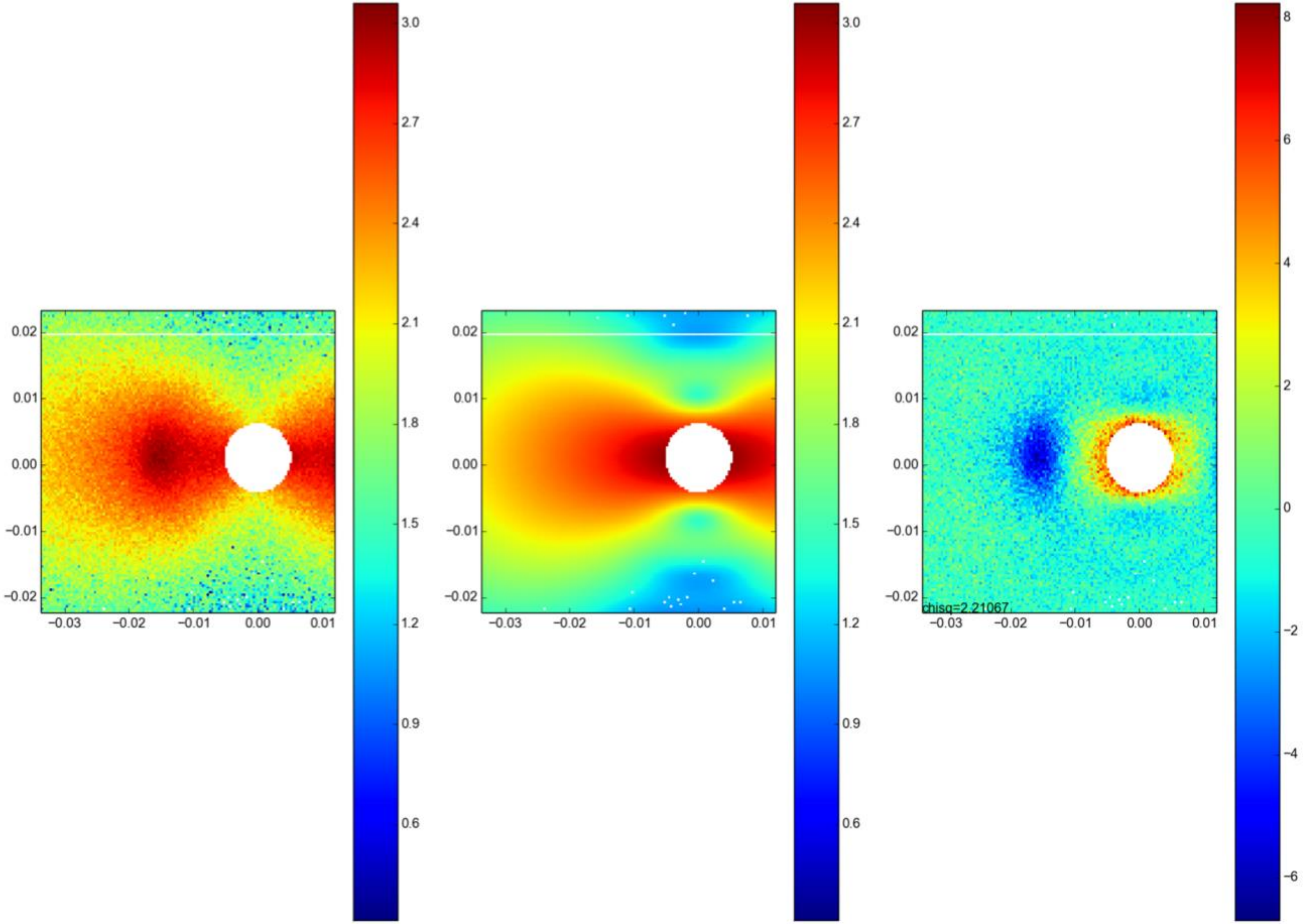
RESULTS

Model	Sasview	GPU	Speedup
Cylinder	3977.7ms	202.3ms	19.7X
Ellipse	2953.2ms	285.5ms	10.3X
Core-Shell-Cylind	71149.9ms	4474.7ms	15.9X
Triaxial-Ellipse	100627ms	6500.2ms	15.5X
Lamellar	69.2ms	6.2ms	11.2X

RESULTS

- **Day-long fit to hour-long fit**
- Paul: **50X** faster—cuts out values when the polydispersity weight is low, use local memory
- If 4 GPUs: 4 times faster (200X)
- Allows increased control over simultaneous fitting, multiple-model fitting, and the angular limits of integration in 1D
- Also used models to fit various scattering data

- Here is an example of a fit for a surfactant at 0 Hz
- The left is the data, the middle is the fit, and the right is the residuals of the data



ACKNOWLEDGEMENTS

- Paul Kienzle
- Dr. Matt Wasbrough
- Aaron West
- Yusuf Ameri cainaki

OPENCL VS CUDA

CUDA:

- Easier to understand; more tutorials online and in books
- CUDA: need to have whole toolchain available

OpenCL:

- Newer, so not much online; less accessible to learn
- Broader range of hardware supported
- simply link the shared library to access